Mihir Dalal
2019EEB1171

CS201- Data Structures Report

ANALYSIS OF DIFFERENT TYPES OF HEAPS

THEORY

1) Array Based implementation of Dijkstra's Algorithm:

Array based implementation of Dijkstra involves a vector of pairs in which the first value consists of distance and the second value consists of the node. For finding the node with minimum first value in our vector, the best complexity is O(size of vector)(if the vector is unsorted). The overall complexity in Dijkstra's array based implementation is O(V^2) where V is the number of vertices and E is the number of Edges in our graph. As the Bellman Ford algorithm's time complexity is O($V * E$), the complexity of Johnson's Algorithm turns out to be O($V^3 + V * E$)(this is because Dijkstra's Algorithm runs V times).

2) Binary Heap based implementation of Dijkstra's Algorithm:

Binary heap implementation of Dijkstra's Algorithm is faster than the Array based implementation and the complexity of Dijkstra's algorithm using Binary Heaps is O($(V+E) * log(V)$) which is better than array based implementation. As the Bellman Ford algorithm's time complexity is O($V * E$), the complexity of Johnson's Algorithm turns out to be O(

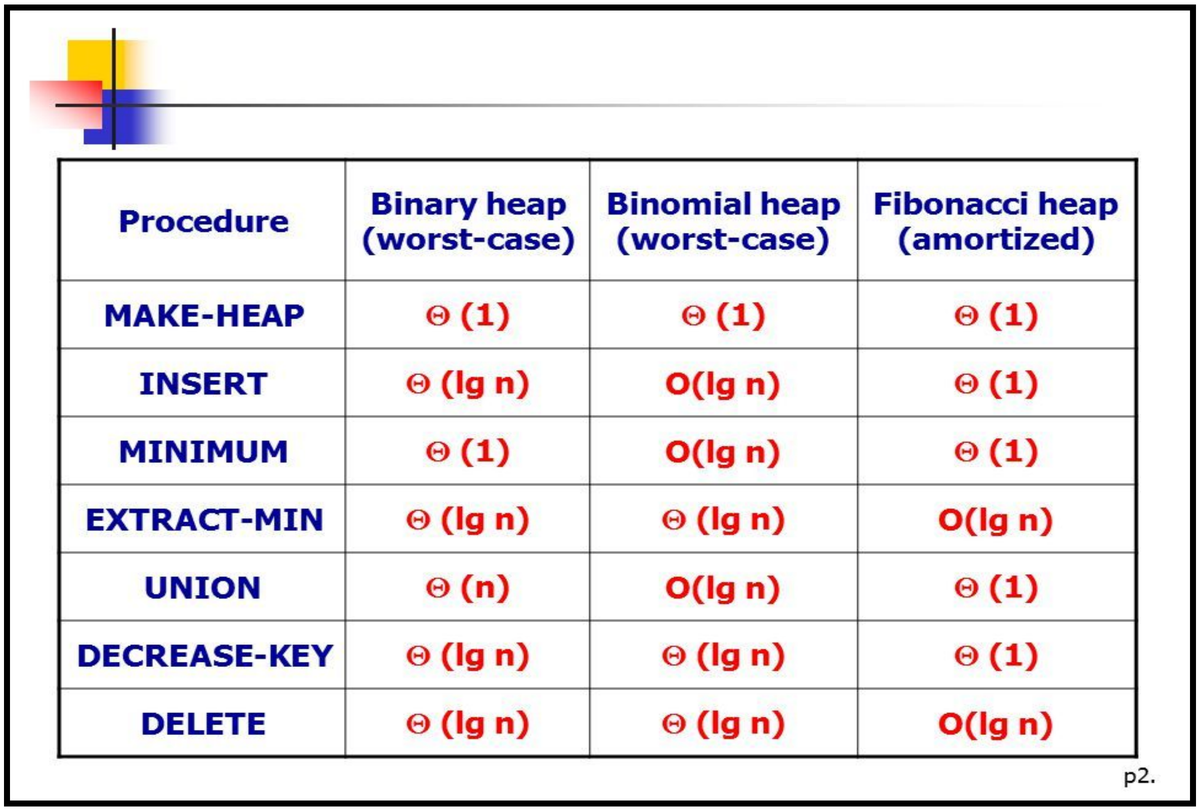
$V^2 + V * E * log(V)$)(this is because Dijkstra's Algorithm runs V times). The insert function in binary heap has O($log\ n$) complexity, the decreasekey operation has complexity O($log\ n$), the extract min function has complexity of O($log\ n$) and the delete key function also takes O($log\ n$) complexity.

3) **Binomial Heap based implementation of Dijkstra's Algorithm:**

Binomial heap has a slightly better performance as compared to binary heap when it comes to union function which takes O($log\ n$) complexity as compared to O(n) of binary heap union. But, finding minimum node in binary heap is O(1)which is better than O($log\ n$) of binomial heap. The complexity of Johnson's Algorithm using Binomial heap in Dijkstra's Algorithm is O($V^2 + V * E * log(V)$).

4) **Fibonacci Heap based implementation of Dijkstra's Algorithm:** Fibonacci heap based implementation of Dijkstra's algorithm is the most efficient implementation as compared to all of the above types of heaps. This is a great improvement from the above three heaps as the operations take O($log\ n$) time complexity only in the functions of extract min and delete key (amortized complexities). Rest all other functions work in O(1) time complexity. Here, we use circular doubly linked list to implement our fibonacci heap. We also maintain a pointer to the lowest value to get the minimum value in O(1) time complexity. The time complexity of Johnson's algorithm, using Fibonacci heaps in the implementation of Dijkstra's algorithm, is O($V^2 log(V) + V * E$).

The diagram below compares the time complexities of different types of heaps :-



Procedure	Binary heap (worst-case)	Binomial heap (worst-case)	Fibonacci heap (amortized)
MAKE-HEAP	Θ (1)	Θ (1)	Θ (1)
INSERT	Θ (lg n)	O(lg n)	Θ (1)
MINIMUM	Θ (1)	O(lg n)	Θ (1)
EXTRACT-MIN	Θ (lg n)	Θ (lg n)	O(lg n)
UNION	Θ (n)	O(lg n)	Θ (1)
DECREASE-KEY	Θ (lg n)	Θ (lg n)	Θ (1)
DELETE	Θ (lg n)	Θ (lg n)	O(lg n)

p2.

Fig: Comparing complexities of different types of heaps.

Source: https://slideplayer.com/slide/5187292/

Analysis of Different types of heaps on Random- Large Graphs *

Here, V is the number of vertices, E is the number of edges and d=0 if undirected and d=1 if the graph is directed.

1) **V=500 and E=1500 and d=1**
    - **Runtime of Array implementation:** 0.650580 seconds
    - **Runtime of Binary Heap implementation:** 0.536619 seconds
    - **Runtime of Binomial Heap implementation:** 0.102779 seconds

* As there is a segmentation fault in my code for fibonacci heap, I am unable to do analysis for fibonacci heap.

- **Analysis:** Here, we see that Binomial heap implementation takes the least amount of time to execute as expected followed by Binary heap and lastly followed by Array based implementation.

2) V=600 and E=1500 and d=1

- **Runtime of Array implementation:** 0.64702 seconds
- **Runtime of Binary Heap implementation:** 0.625688 seconds
- **Runtime of Binomial Heap implementation:** 0.244334 seconds

- **Analysis:** Here, we see that Binomial heap implementation takes the least amount of time to execute as expected followed by Binary heap based implementation and lastly followed by Array based implementation.

3) V=750 and E=2000 and d=1

- **Runtime of Array implementation:** 0.137433 seconds
- **Runtime of Binary Heap implementation:** 0.947593 seconds
- **Runtime of Binomial Heap implementation:** 0.267187 seconds

- **Analysis:** Here, we see that Array implementation unexpectedly jumps to the top followed by binomial heap followed by Binary Heap.

4) V=800 and E=1000 and d=0

- **Runtime of Array implementation:** 0.08454 seconds
- **Runtime of Binary Heap implementation:** 0.0812873 seconds
- **Runtime of Binomial Heap implementation:** 0.06944 seconds

- **Analysis:** Here, we see that Binomial heap implementation takes the least amount of time to execute as expected followed by

Binary heap and lastly followed by Array based implementation. So in this case of undirected graph too, Binomial heap is the clear winner.

5) <u>V=1000 and E=1000 and d=0</u>

- **Runtime of Array implementation:** 0.024001 seconds
- **Runtime of Binary Heap implementation:** 0.122873 seconds
- **Runtime of Binomial Heap implementation:** 0.021453 seconds
- **<u>Analysis:</u>** Here, we see that Binomial heap implementation takes the least amount of time to execute as expected followed by Array based implementation and lastly followed by binary heap. Binomial heap, as usual is the winner here.

6) <u>V=1000 and E=1000 and d=1</u>

- **Runtime of Array implementation:** 0.018546 seconds
- **Runtime of Binary Heap implementation:** 0.111708 seconds
- **Runtime of Binomial Heap implementation:** 0.017951 seconds
- **<u>Analysis:</u>** Here, we see that Binomial heap implementation takes the least amount of time to execute as expected followed by Array based implementation and lastly followed by binary heap. Binomial heap, as usual is the winner here.

<u>Observations:</u> We can observe that in many cases binary based implementation outshines the binary heap based implementation and in the rest of the cases array based implementation outperforms binary heap based implementation. This may have occurred due to some implementation ambiguities. By theory, we know that fibonacci heap is the fastest of them all. As my code for fibonacci heap is not in working condition, we cannot check the running time for fibonacci heap.

We observe here that Binomial heap always outperforms Array based implementation and binary heap based implementations. So, Binomial heap is the clear winner amongst them. However, we have that fibonacci heap is the fastest among all.

<u>**Conclusion:**</u> We now conclude that the time taken by various heaps in Johnson's Algorithm (in Dijkstra's Algorithm) can be figured out as follows:

Fibonacci Heap < Binomial Heap < Binary Heap < Arrays

We have tested the above order on random large graphs. Hence, we may conclude that fibonacci heaps are the most efficient heaps among all of the above heaps, followed by Binomial heap, followed by binary heaps and then the most inefficient type that is Array based implementation.

****